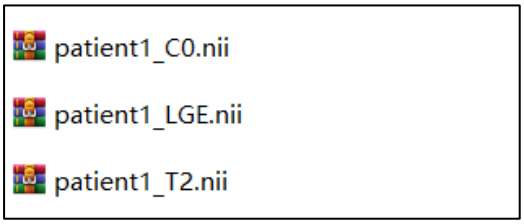
cmr 方位自动调整实验报告

郁晨阳

一、数据集准备

首先需要在网站 https://zmiclab.github.io/zxh/0/mscmrseg19/data.html 申请下载数据集，下载到的数据当中并非已经含有全部方位，里面共 45 位病人的心脏磁共振图像信息，每个病人都有三种心脏磁共振图像文件，都含有若干切片，每个图像文件打开后方位是固定的，但不是校准好的方位，因此要先对每个文件进行方位的判断，即先把所有的原始数据标上方位标签，再由原始数据生成其他 7 个方位的图像数据。

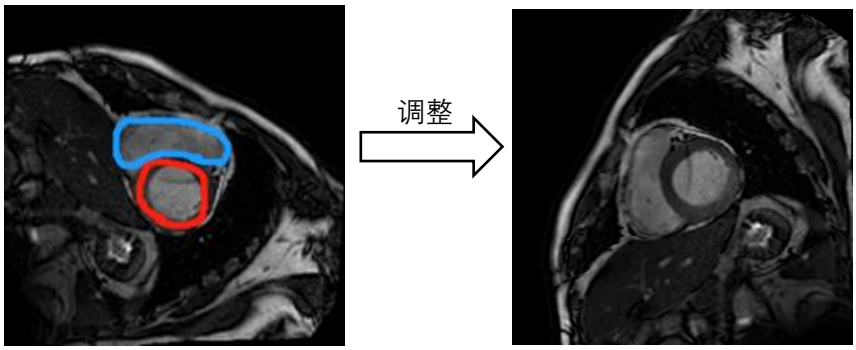


关于方位标签 000~111 具体含义，请看下表：

标签	含义	标签	含义
000	正确方位图像	100	由正确方位沿“\”翻转
001	由正确方位水平翻转	101	由正确方位顺时针旋转 90°
010	由正确方位竖直翻转	110	由正确方位顺时针旋转 270°
011	由正确方位顺时针旋转 180°	111	由正确方位沿“/”翻转

我们的目标是让神经网络能自动识别输入的心脏磁共振图像方位，并转换成 000 方位。

首先是对原始数据进行方位标注，那么要先明确 000 到底是什么样子的，如下方左图，我们选取某位病人文件，选择中间的切片显示出来进行方位判断，这是因为第一个或者最后一个拍到的都是心脏边缘的形状，可能会看不清楚。可以观察到，左心室（红笔圈出来的部分）在下方，右心室（蓝笔圈出来的部分）在上方，那么这个方位就不是校准好的，我们想要的正确方位图像应该是左心室在右边，右心室在左边，后背、肝脏在下方，如下方右图所示。



可以看到左图可以通过右图（000）顺时针旋转 90°得到，因此左图应标记为 101。通过人工查看所有图像发现，下载到的数据集标签都是 101，下面就可以着手准备所有方位的数据集了。

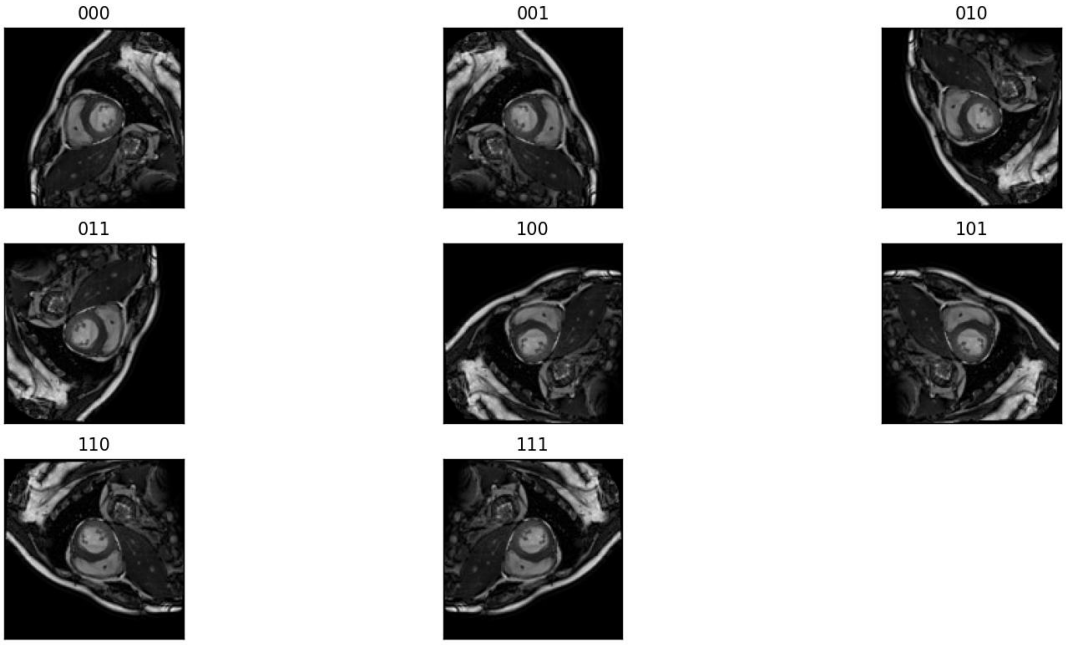
将所有 nii 文件读取到的三维数据的所有切片下采样至 128*128，目的是减小神经网络输入数据的大小，再全部保存至 csv 文件中，保存的是 numpy.ndarray 类型二维数据，利用简单的线性变化获得其余 7 种方位的 cmr 图像（都是单通道灰度图）存至 csv 文件中。同时将方位标签添加在每个文件名的最末。在后续搭建 UI 用户界面的时候，需要用到以下处理方式的逆向操作对 cmr 图像的方位进行校准。

注意下表展示的是通过原始数据标签为 000 的原始数据得到其他 7 个方位数据的具体方法，本实验原始数据是 101，那么处理过程中会稍有不同。

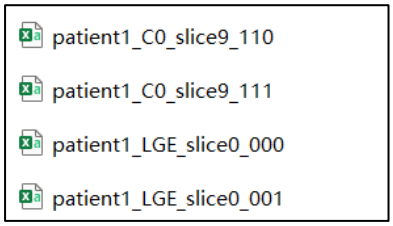
方位标签	处理方式	方位标签	处理方式
000	origin_data	100	origin_data .transpose()
001	origin_data [:, ::-1]	101	origin_data .transpose()[:,::-1]
010	origin_data [::-1, :]	110	origin_data [:, ::-1].transpose()
011	origin_data [::-1, ::-1]	111	origin_data [::-1, ::-1].transpose()

其中：orgin_data 表示原始二维数据

最终同一幅图像的 8 种方位如下图所示可以结合第一页的方位标签含义表一起看：



在给神经网络准备数据的时候直接通过获取文件内的 numpy 数据再转成 tensor 类型得到图像数据信息，获取文件名最末的三位二进制数转换成十进制数来得到标签信息。



将数据分成训练集和测试集两份，选病人 1 号至 40 号作为训练集，41 号到 45 号作为测试集，分别存放在两个文件夹 train 和 test 中，其中训练集的长度为： 9856，测试集的长度为：1228，大致为 8：1。训练过程中，epoch 设置为 10，batch_size 设置为 64。

准备数据集这一部分的代码可以在以下两个文件中查看
prepare_mydata.py：生成 8 种方位的 csv 文件
load_my_data.py：继承 Dataset 类，为输入神经网络做准备

prepare_mydata.py 的部分代码截图如下：

```
# 001图像
csv_dir_path = "D:\\workfile\\pytorch_project\\pro1\\cmrdata"
all_csv_name = os.listdir(csv_dir_path)
for one_csv_name in all_csv_name:
    one_csv_path = os.path.join(csv_dir_path, one_csv_name)
    one_csv_data = np.loadtxt(one_csv_path)
    one_csv_data = one_csv_data[:, ::-1]
    np.savetxt("D:\\workfile\\pytorch_project\\pro1\\labeled_cmr_cvs_001\\{}_slice{}_001.csv".
               format(one_csv_name[:one_csv_name.find(".")], one_csv_name[-5:-4]), one_csv_data)
```

load_my_data.py 代码截图如下：

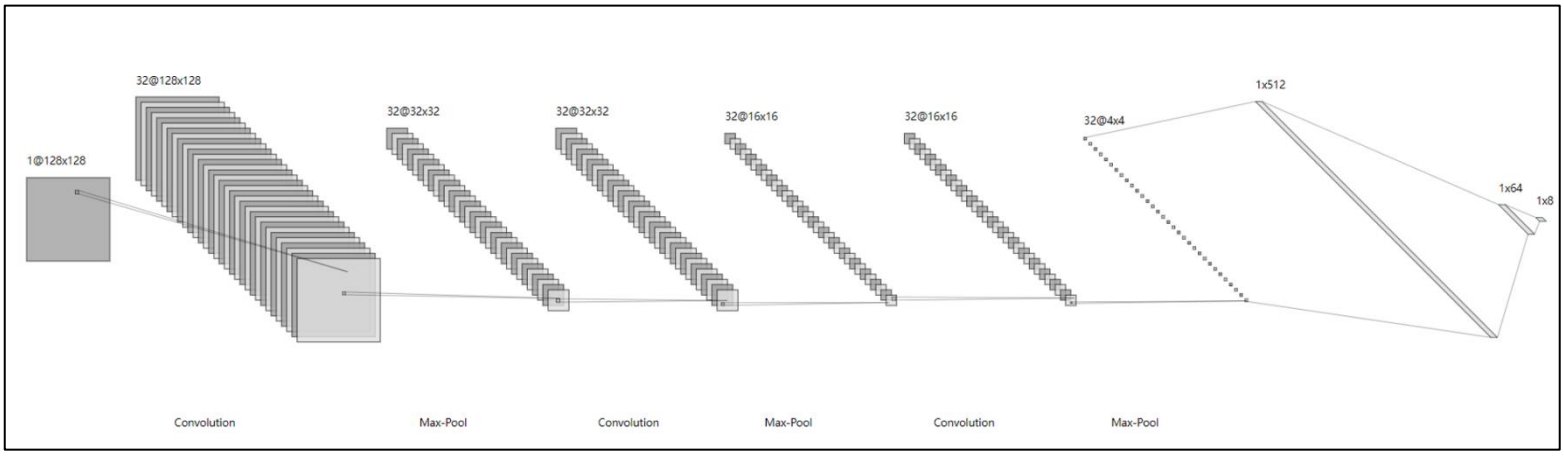
```
from torch.utils.data import Dataset
import os
import numpy as np
from torchvision import transforms


class MyData(Dataset):
    def __init__(self, img_dir):
        self.img_dir = img_dir
        self.all_img_name = os.listdir(self.img_dir)

    def __getitem__(self, index):
        img_name = self.all_img_name[index]
        img_path = os.path.join(self.img_dir, img_name)
        img = np.loadtxt(img_path).astype(np.float32)
        img_lab = int(img_name[-7:-6])*4+int(img_name[-6:-5])*2+int(img_name[-5:-4])
        trans_totensor = transforms.ToTensor()
        img_tensor = trans_totensor(img)
        return img_tensor, img_lab

    def __len__(self):
        return len(self.all_img_name)
```

二、卷积神经网络设计与训练



总体上有 3 个卷积层，再加上 2 个线性层，其中每一个卷积层 Conv 和最大池化层 MaxPool 中间都添加批标准化 BatchNorm 和激活层 ReLU。

详细代码如下:

```python
import torch
from torch import nn


# 搭建神经网络
class MyCnn(nn.Module):
    def __init__(self):
        super(MyCnn, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(1, 32, 5, 1, 2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(4),
            nn.Conv2d(32, 32, 5, 1, 2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(4),
            nn.Conv2d(32, 32, 5, 1, 2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(32*4*4, 32),
            nn.Linear(32, 8)
        )

    def forward(self, x):
        x = self.model(x)
        return x
```

损失函数选择 Pytorch 常用的交叉熵损失函数 CrossEntropyLoss，优化器选择随机梯度下降 SGD，学习率设置为 0.01。

```
# 损失函数
loss_fun = nn.CrossEntropyLoss()
loss_fun = loss_fun.to(device)

# 优化器
learning_rate = 0.01
optimizer = torch.optim.SGD(my_cnn.parameters(), lr=learning_rate)

# 设置训练网络的一些参数
# 训练轮数
epoch = 10
```

每完成一轮训练（epoch），对训练集和测试集方位判断的正确率进行统计，并通过 run 窗口打印出来，下面是某次训练的结果：

```
训练集的长度为：9856
测试集的长度为：1224
第 1 轮训练开始
整体训练集上的正确率：0.8547077775001526
整体测试集上的正确率：0.9967320561408997
第 2 轮训练开始
整体训练集上的正确率：0.9956371784210205
整体测试集上的正确率：1.0
第 3 轮训练开始
整体训练集上的正确率：0.9991883039474487
整体测试集上的正确率：1.0
第 4 轮训练开始
整体训练集上的正确率：0.9995941519737244
整体测试集上的正确率：1.0
第 5 轮训练开始
整体训练集上的正确率：0.9997971057891846
整体测试集上的正确率：1.0
第 6 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 7 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 8 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 9 轮训练开始
整体训练集上的正确率：1.0
```

整体测试集上的正确率：1.0
第 10 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0

可以看到第一轮的正确率测试集甚至好于训练集，训练多次，每次第一轮的正确率都是测试集上更高，另外就是第二轮开始无论训练集还是测试集，正确率都相对较高，分析原因，可能是数据量偏少，虽然每个 nii 文件都有很多切片，但切片之间相似度比较高的，从而虽然看上去有上万个数据，其实还是很少，只有 45 位病患图像。再加上不同方位的图像本身是通过线性变换生成，比较简单，不像实际情况那样复杂，因此神经网络很快就过拟合了。

防止偶然性，将 1 到 5 号病人作为测试集，6 到 45 号病人作为训练集，训练结果如下：

训练集的长度为：9840
测试集的长度为：1240
第 1 轮训练开始
整体训练集上的正确率：0.8189024329185486
整体测试集上的正确率：0.9919354915618896
第 2 轮训练开始
整体训练集上的正确率：0.987093448638916
整体测试集上的正确率：0.999193549156189
第 3 轮训练开始
整体训练集上的正确率：0.9972561001777649
整体测试集上的正确率：1.0
第 4 轮训练开始
整体训练集上的正确率：0.9989837408065796
整体测试集上的正确率：1.0
第 5 轮训练开始
整体训练集上的正确率：0.9996951222419739
整体测试集上的正确率：1.0
第 6 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 7 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 8 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 9 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0
第 10 轮训练开始
整体训练集上的正确率：1.0
整体测试集上的正确率：1.0

训练完一次之后将数据保存在 my_cnn.pth 文件中，因为 UI 界面程序要用到，这个文件就直接放在了 pyqt 项目文件夹中，神经网络设计部分的代码存放在 MyCnn.py 文件中，训练部分的代码存放在 train_cmr.py 文件当中。

完整的代码截图如下：

```python
from torch.utils.data import DataLoader
from MyCnn import *
from load_my_data import MyData


# 定义训练的设备
device = torch.device("cuda")

# 准备数据集
train_dir = "Address Of Train Data"
test_dir = "Address Of Test Data"
train_data = MyData(train_dir)
test_data = MyData(test_dir)

# 获取数据集的长度
train_data_size = len(train_data)
test_data_size = len(test_data)

# 如果train_data_size=10，则输出：训练集的长度为：10
print("训练集的长度为：{}".format(train_data_size))
print("测试集的长度为：{}".format(test_data_size))

# 利用dataloader加载数据集
train_data_loader = DataLoader(train_data, batch_size=64)
test_data_loader = DataLoader(test_data, batch_size=64)

# 创建网络模型
my_cnn = MyCnn()
my_cnn = my_cnn.to(device)
```

```python
# 损失函数
loss_fun = nn.CrossEntropyLoss()
loss_fun = loss_fun.to(device)

# 优化器
learning_rate = 0.01
optimizer = torch.optim.SGD(my_cnn.parameters(), lr=learning_rate)
```

```python
# 设置训练网络的一些参数
# 训练轮数
epoch = 10
# 记录训练次数
total_train_step = 0
# 记录测试的次数
total_test_step = 0


for i in range(epoch):
    print("第{}轮训练开始".format(i+1))

    # 训练开始
    total_train_accuracy = 0
    for data in train_data_loader:
        imgs, targets = data
        imgs = imgs.to(device)
        targets = targets.to(device)
        # print(imgs.shape)
        outputs = my_cnn(imgs)
        targets = torch.as_tensor(targets)
        loss = loss_fun(outputs, targets)
        train_accuracy = (outputs.argmax(1) == targets).sum()
        total_train_accuracy = total_train_accuracy + train_accuracy
        # 优化器优化模型
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_train_step += 1
        # if total_train_step % 100 == 0:
        #     print("训练次数：{}，loss：{}".format(total_train_step, loss))
    print("整体训练集上的正确率：{}".format(total_train_accuracy/train_data_size))

    # 测试开始
    total_test_loss = 0
    total_test_accuracy = 0
    with torch.no_grad():
        for data in test_data_loader:
            imgs, targets = data
            imgs = imgs.to(device)
            targets = targets.to(device)
            outputs = my_cnn(imgs)
            targets = torch.as_tensor(targets)
            loss = loss_fun(outputs, targets)
            total_test_loss = total_test_loss + loss.item()
            accuracy = (outputs.argmax(1) == targets).sum()
            total_test_accuracy = total_test_accuracy + accuracy
    # print("测试集loss:{}".format(total_test_loss))
    print("整体测试集上的正确率：{}".format(total_test_accuracy/test_data_size))

# 保存训练完的神经网络参数数据
torch.save(my_cnn.state_dict(), "pyqt_class\\my_cnn.pth")
```

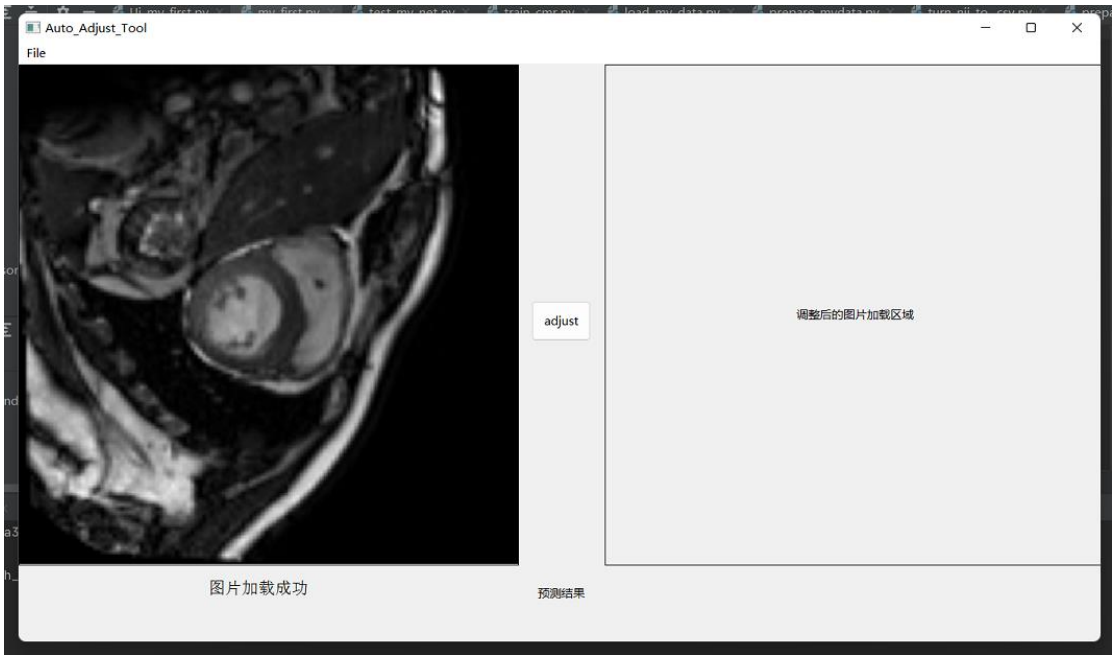
三、用户界面设计

主要功能有两个，一个是打开并显示心脏磁共振图像，另一个是设置一个按钮来调整载入的心脏磁共振图像方位，如果校准成功，则通过文本来提示。

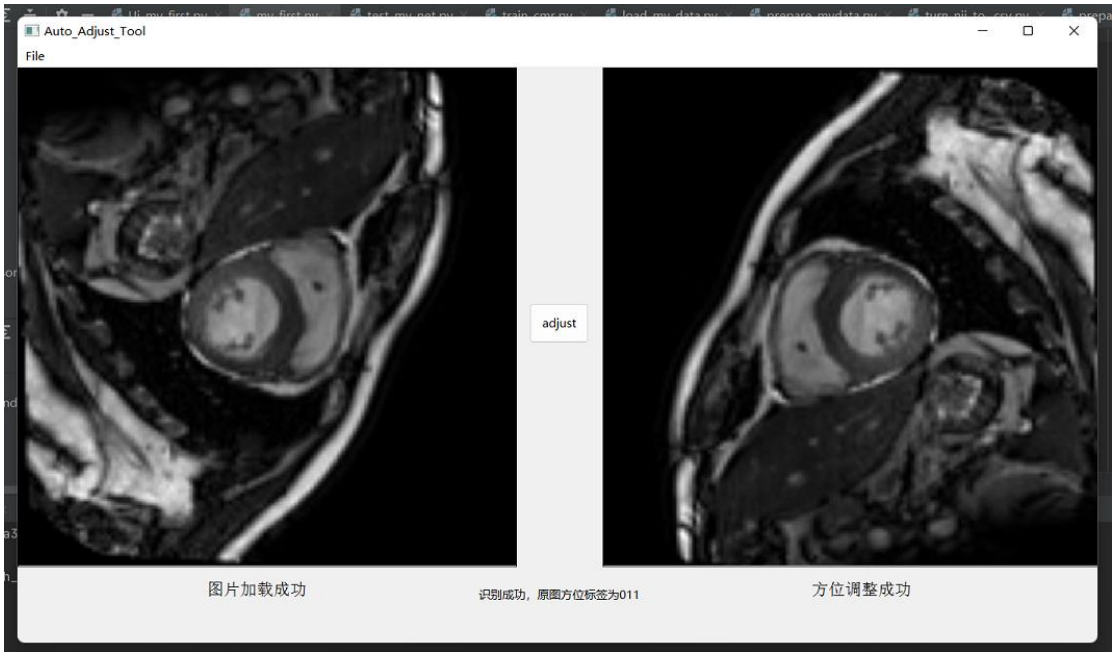
采用的工具包是 pyqt6，利用 qt designer 设计 UI 界面，并借助了 eric7 来自动编译生成 py 文件，这样省去了大量的熟悉 pyqt6 包中各种类的时间，只需要在相应的事件触发函数中填补上功能代码即可。

整体的界面如下，显示文本和图片的组件都用了 Qlabel。

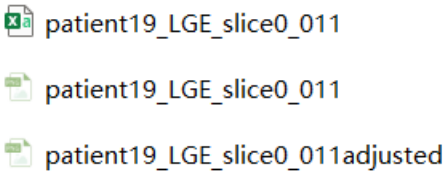
这里先简单说明一下使用效果，点击菜单栏 File 可以弹出两个按钮 open 和 save，(save 暂时没有给它赋上保存文件的功能)，点击 open 即可选择一个 csv 文件(在 labeled_cmr_cvs 问件夹中)，将图像信息显示在原图加载区域，并在图像下面显示加载成功的提示，如下图:

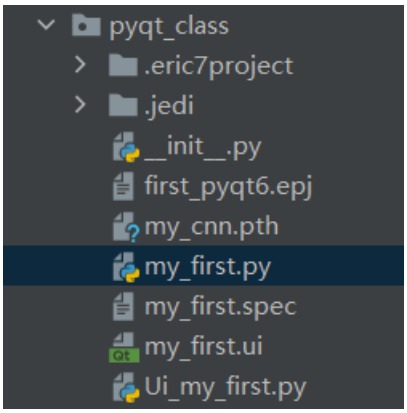


接下来可以点击 adjust 按钮对图像进行方位校准，窗口下方的“预测结果”会显示判断方位有没有成功，并显示方位标签，之后显示校准后的图片，并在图片下面显示调整成功的提示（只说明做了调整方向的操作，具体有没有成功要看 adjust 按钮下方的文本提示），如下图：



由于对 pyqt 的组件还不太熟悉，这里还存在着一些问题， Qlabel 显示图片需要直接给它一个图像文件，所以目前每加载一个 csv 文件都会在当前文件夹中生成一个 png 图片，每 adjust 一下，都会在当前文件夹中生成一个对应的调整好方位的 png 图片。但是每次操作只会在当前文件夹中生成一个图，不会越点越多，如下图：





最后如上图所示，是 pyqt 文件夹中的一些文件，这里简单说明，my_first.ui 文件是用 eric7 直接创建的，创建完就可以在 qt designer 中添加各种组件，设计好之后直接用 eric7 编译，生成两个 py 文件，Ui_my_first.py 和 my_first.py，前一个是生成 UI 界面的主文件，包含主窗口类（不清楚用语对不对），后一个继承了前面一个文件中的主窗口类，内部有按钮按下的事件函数，因此在实现相应功能的时候，第一个文件不要修改，只在第二个文件中添加功能代码，最终使用软件时也是运行 my_first.py 代码。

这里只截图展示功能函数部分的代码：
open 按钮打开并显示一个 cmr 图像

```
@pyqtSlot()
def on_actionopen_triggered(self):
    """
    Slot documentation goes here.
    """
    # TODO: not implemented yet
    try:
        file_name, file_type = QFileDialog.getOpenFileNames(self, "选取文件", "./",
                                                            filter="csv Files (*.csv);;All Files (*)")
        print("文件名：")
        global global_file_path
        global_file_path = ''.join(file_name)
        print(global_file_path)
        one_csv_data = np.loadtxt(global_file_path)
        new_size = (512, 512)
        one_csv_data = cv.resize(one_csv_data, new_size)
        global_file_path = global_file_path[:-3] + 'png'
        mplimg.imsave(global_file_path, one_csv_data, cmap='gray')
        pix = QPixmap(global_file_path)
        print("pix is ok")
        self.label_ori.setPixmap(pix)
        self.label_tip.setText("图片加载成功")
    except:
        pass
```

adjust 按钮自动调整方位

```
@pyqtSlot()
def on_pushButton_2_clicked(self):
    """
    Slot documentation goes here.
    """
    # TODO: not implemented yet
    # print(global_file_path)
    global global_file_path
    global_file_path = global_file_path[:-3] + 'csv'
    img_lab = int(global_file_path[-7:-6]) * 4 + int(global_file_path[-6:-5]) * 2 + int(global_file_path[-5:-4])
    print(img_lab)
    # 准备输入神经网络的数据
    img_data = np.loadtxt(global_file_path).astype(np.float32)
    trans_totensor = transforms.ToTensor()
    img_tensor = trans_totensor(img_data)
    # print(img_tensor.shape)
    img_tensor = torch.reshape(img_tensor, (1, 1, 128, 128))
    # 创建网络模型
    my_cnn = MyCnn()
    my_cnn.load_state_dict(torch.load(("my_cnn.pth")))
    # print(my_cnn)
```

```
    output = my_cnn(img_tensor)
    label = output.argmax(1).item()
    label_dict = {0: "000", 1: "001", 2: "010", 3: "011", 4: "100", 5: "101", 6: "110", 7: "111"}
    if label == img_lab:
        self.label_rel.setText("识别成功，原图方位标签为{}".format(label_dict[label]))
    else:
        self.label_rel.setText("识别失败")
    one_csv_data = adjust_tool(label, img_data)
    new_size = (512, 512)
    one_csv_data = cv.resize(one_csv_data, new_size)
    global_file_path = global_file_path[:-4] + 'adjusted.png'
    mplimg.imsave(global_file_path, one_csv_data, cmap='gray')
    pix = QPixmap(global_file_path)
    # print("pix is ok")
    self.label_adj.setPixmap(pix)
    self.label_tip_2.setText("方位调整成功")
```

其中圈出来的调用的 adjust_tool 函数如下：

```
def adjust_tool(label, one_csv_data):
    if label == 0:
        return one_csv_data
    if label == 1:
        one_csv_data = one_csv_data[:, ::-1]
        return one_csv_data
    if label == 2:
        one_csv_data = one_csv_data[::-1, :]
        return one_csv_data
    if label == 3:
        one_csv_data = one_csv_data[::-1, ::-1]
        return one_csv_data
    if label == 4:
        one_csv_data = one_csv_data.transpose()
        return one_csv_data
    if label == 5:
        one_csv_data = one_csv_data[:, ::-1].transpose()
        return one_csv_data
    if label == 6:
        one_csv_data = one_csv_data.transpose()[:, ::-1]
        return one_csv_data
    if label == 7:
        one_csv_data = one_csv_data.transpose()[::-1, ::-1]
        return one_csv_data
```

四、实验总结

整个实验还是花了较多的时间完成的，准备数据集、熟悉 pytorch 框架搭建神经网络、用 pyqt 设计 UI 界面，虽然很简单，但是中间还是遇到了不少的困难，尤其在数据集上面，花了相对较多的时间，也因此意识到了数据的重要性，“种瓜得瓜，种豆得豆”，只有输入良好的数据，才可能有良好的结果。另一个收获是锻炼了自己写代码和解决 bug 的能力，虽然都是一些很简单的问题，但对我来说也是一个小小的进步。

当然小程序问题还是比较多的，例如图片显示带来的自动保存文件、正确率很快收敛到1、难以分析神经网络训练的效果好坏、过拟合问题等等，需要在后续的实验中进一步完善。